

KARELIA-AMMATTIKORKEAKOULU  
Tietojenkäsittelyn koulutusohjelma

Janne Turunen

Monen pelaajan HTML5-autopeli Node.js:llä

Opinnäytetyö  
Toukokuu 2014



**OPINNÄYTETYÖ**  
**Toukokuu 2014**  
**Tietojenkäsittelyn koulutusohjelma**

Karjalankatu 3  
80260 JOENSUU  
Puh. 013 260 6800

**Tekijä(t)**  
Janne Turunen

**Nimeke**  
Monen pelaajan HTML5-autopeli Node.js:llä

**Toimeksiantaja**  
Karelia-ammattikorkeakoulu

**Tiivistelmä**

Tämän opinnäytetyön toimeksiantajana toimi Karelia-ammattikorkeakoulu. Työssä käydään läpi pelinkehitysprosessi, jossa suunnitellaan ja kehitetään reaaliaikainen monelle pelaajalle suunnattu HTML5-verkkopeliprototyyppi. Reaaliaikaisen moninpelin kehittäminen tuo pelinkehitykseen omat haasteensa, joita työssä myös käsitellään.

Opinnäytetyön käytännön osuudessa toteutettiin monen pelaajan 2D-autopeliprototyyppi, jota pystyy pelaamaan selaimessa verkon välityksellä. Opinnäytetyössä käsitellään monia pelin tekemiseen liittyviä osa-alueita kuten pelin suunnittelua, ohjelmointia ja julkaisua.

Teoriaosuudessa käsitellään Node.js-sovellusallustaa, jonka avulla saadaan käsiteltyä JavaScript-koodia palvelinpuolella. Työssä esitellään myös muita pelinkehitysprosessin aikana käytettyjä työkaluja, kuten OpenShift-pilvialusta, jossa peli on toiminnassa. Itse pelin toiminnallisuus tehtiin Phaser.js-kirjastolla ja JavaScript-ohjelmointikielellä.

Lopputuloksena syntyi yksinkertainen nettisivusto, jossa peliprototyyppi on toiminnassa ja jossa pelaajat pääsevät ajamaan yksittäisiä kisoja toisia vastaan.

**Kieli**  
suomi

**Sivuja** 39

**Asiasanat**  
verkkopeli, moninpeli, HTML5-peli, Node.js



**THESIS**  
**May 2014**  
**Degree Programme in Computer Science**  
Karjalankatu 3  
FI 80260 JOENSUU  
FINLAND  
Tel. +358 13 260 6800

Author(s)  
Janne Turunen

Title  
Multiplayer car game in HTML5 with Node.js

Commissioned by  
Karelia University of Applied Sciences

#### Abstract

The client of this thesis was Karelia University of Applied Sciences. Thesis goes through a process of planning and developing a prototype of a real-time multiplayer HTML5 game that works in internet. Developing a real-time multiplayer game brings its own challenges that are also discussed in thesis.

In the practical part of the thesis a multiplayer 2D car game prototype was developed that can be played in browser via internet. Thesis discusses about many fields of game developing such as planning, programming and publishing.

The theoretical part discusses about Node.js application platform that can be used to handle JavaScript code server-side. Thesis also discusses about other tools used in the process such as OpenShift cloud platform where the game is running. The functionality of the game was made with Phaser.js library and JavaScript programming language.

The end result was a simple webpage, where the game prototype is running and where players can drive single races against each other.

Language  
Finnish

Pages 39

Keywords  
network game, multiplayer game, HTML5 game, Node.js

# Sisältö

## Termit ja lyhenteet

1	Johdanto.....	7
2	Node.js .....	9
2.1	Noden historia.....	11
2.2	Npm .....	12
2.3	Projektissa käytetyt Node-paketit.....	13
2.3.1	Express .....	13
2.3.2	Socket.io .....	14
2.3.3	Passport.....	15
2.3.4	Mongoose .....	16
3	HTML5-pelin kehittäminen.....	17
3.1	HTML5-pelimootorit .....	18
3.1.1	Phaser.....	19
3.2	Fysiikan mallintaminen.....	21
3.3	Ratojen suunnittelu .....	24
3.4	Grafiikat .....	25
3.5	Äänet .....	27
3.5.1	HTML5 Audio .....	27
3.5.2	Web Audio.....	28
3.6	Verkkopelin kehittämisen haasteet .....	29
3.6.1	Huijaaminen .....	30
3.6.2	Latenssi.....	30
3.6.3	Asiakaspään ennakointi .....	31
3.6.4	Dead reckoning .....	31
3.7	Verkkosivuston kehittäminen ja toiminta .....	32
3.8	Pelin ja sivuston julkaisu .....	34
3.8.1	OpenShift-pilvipalvelu.....	34
4	Pohdinta .....	35
4.1	Kehittämisideat .....	36
	Lähteet.....	38

## Termit ja lyheneet

Asynkronisuus	Katso Non-blocking I/O.
Debuggaus	Menetelmä, jolla sovelluksesta etsitään ja poistetaan mahdollisia virheitä.
DOM	Document Object Model, rajapinta, jonka avulla sovellukset voivat dynaamisesti käsitellä ja päivittää HTML-dokumenttien sisältöä, rakennetta ja tyyliä.
Domain	Verkkotunnus, esimerkiksi <a href="http://www.esimerkki.fi">www.esimerkki.fi</a> .
DOS	Disk Operating System, IBM-yhteensopivien tietokoneiden komentorivipohjainen käyttöjärjestelmäperhe, jonka ensimmäinen versio julkaistiin vuonna 1981.
Flash	Adobe Systemsin tuottama alusta, jonka avulla verkkosivuilta saadaan esitettyä multimediasisältöä.
Frame	Kuvat, näytetään sarjana niin, että liikkeestä tulee sulavan näköistä. Fps (frames per second) eli kuvaa sekunnissa on yleinen merkintätapa ruudunpäivitysnopeudelle peleissä.
HTML5	HyperText Markup Language, merkkauskieli, jonka avulla verkkosivujen sisällöt rakennetaan.
I/O	Input/Output, tiedonvälitys tietokoneen ja muiden laitteiden välillä.
iOS	Applen kehittämä käyttöjärjestelmä iPhone-, iPod touch-, iPad- ja Apple TV -laitteille.

JSON	JavaScript Object Notation, JavaScriptistä johdettu tiedonvälitysmuoto, jota yleisimmin käytetään tiedon välittämiseen palvelimen ja verkkosovelluksen välillä.
Linux	Käyttöjärjestelmä, jonka nimi tulee Linux-ytimeistä ja jonka suomalais-amerikkalainen Linus Torvalds alun perin kehitti.
Mac OS X	Applen kehittämä käyttöjärjestelmäperhe Macintosh-tietokoneisiin.
Non-blocking I/O	Sallii koodin ajamisen asynkronisesti eli edessäpäin, ennen kuin edellinen operaatio on valmis.
NoSQL	Not Only SQL, relaatiomallista poikkeava tietokantamalli, joka ei seuraa kiinteästi määriteltyä rakennetta.
PC	Personal Computer, henkilökohtainen tietokone. Koon, kykyjen ja hinnan puolesta yksittäisen henkilön käytettäväksi soveltuva tietokone.
Pelimoottori	Ohjelmistokehys, jonka avulla pelintekijät saavat rakennettua pelejä. Sisältää usein työkalut mm. grafiikoihin, fysiikan mallinnukseen, ääniin ja animointiin.
Renderöinti	Prosessi, jossa mallista tuotetaan kuva.
URL	Universal Resource Locator, yksilöllinen osoite www-sivulle tai muulle verkossa sijaitsevalle resurssille.

# 1 Johdanto

Tässä opinnäytetyössä käydään läpi prosessi, jossa suunnitellaan ja kehitetään monelle pelaajalle suunnattu ja verkossa toimiva HTML5-autopeli. Idean peliin sain ohjaajaltani Jaakko Vanhalalta ja toimeksiantajana toimi Karelia-ammattikorkeakoulu.

Verkkoselaimessa toimivat multimediasovellukset ovat jo kauan olleet yleensä Adobe Flash-tekniikalla toimivia, mutta HTML5-tekniikan myötä tilanne on viime vuosina muuttunut. Verrattaessa HTML5:tä ja Flashia keskenään HTML5:n eduksi voidaan laskea sen kyky toistaa multimediasisältöä verkkosivuilla ilman erillisiä asennettavia lisäosia toisin kuin Flashin kohdalla. HTML5 toimii tietokoneiden lisäksi myös useimmissa mobiililaitteissa ja käyttöjärjestelmissä kuten Applen iOS:ssä, jossa ei ole tukea Flashille. Myös HTML5:n suorituskyky on selkeästi parempi esimerkiksi Linux- ja Mac OS X -käyttöjärjestelmissä. (Emaze 2013.)

Aiheen valinta oli helposti perusteltavissa, koska olen aina ollut kiinnostunut ohjelmoinnista ja erityisesti peliohjelmoinnista ja -suunnittelusta. Minulla on jonkin verran aikaisempaa kokemusta verkkosovellusten ja pelien tekemisestä, mutta reaaliaikaisista verkkopeleistä ja selaimessa pelattavista peleistä ohjelmointikokemusta ei aikaisemmin ole. Täten aihe herätti kiinnostuksen oppia uusia tekniikoita pelien tekemiseen liittyen.

Toteuttamani peliprototyypin idea perustuu pitkälti vanhaan klassikkoon, suomalaiseen Slicks 'n Slide -tietokonepeliin, jonka parissa olen aikoinaan viettänyt useita hetkiä. Slicks 'n Slide on Timo Kauppisen kehittämä ajopeli, jonka ensimmäinen versio julkaistiin vuonna 1993. Kyseessä on ylhäältäpäin kuvattu kaksiulotteinen ajopeli PC-alustalle ja DOS-käyttöjärjestelmälle ja sitä voi samanaikaisesti pelata enintään neljä pelaajaa samalla tietokoneella. Tämän opinnäytetyön tavoitteena oli tuoda Slicks 'n Slide -tyylinen ajopeli nykypäivään ja tuottaa samantyylinen mutta modernimpi versio, jossa useampi pelaaja pys-

tyy pelaamaan peliä samanaikaisesti vastakkain verkon välityksellä omilla tietokoneillaan.

Koska opinnäytetyössä kehitelty peli on toteutettu HTML5:lle, toimii se netiselaimessa siten, että mitään erillistä ohjelmistoa ei tietokoneelle tarvitse asentaa. Tämä tarkoittaa sitä, että peli olisi periaatteessa pelattavissa missä tahansa laitteessa, jossa on tekniset vaatimukset täyttävä verkkoselainohjelmisto, kuten useimmissa mobiililaitteissa tai tietokoneiden eri käyttöjärjestelmissä. Opinnäytetyössä peliä on kuitenkin keskitytty tekemään tietokoneen käyttäjän näkökulmasta eikä siihen ole lisätty esimerkiksi kosketusnäytöisille mobiililaitteille tarvittavia kontrollointimahdollisuuksia.

Pelin teossa käytetyistä tekniikoista ja työvälineistä puhuttaessa Node.js on sovellusalue, jonka avulla palvelinpuolella pystyy käsittelemään JavaScript-ohjelmointikielellä. Noden asynkroninen toimintamalli on erinomainen, kun tehdään reaaliaikaisesti toimivaa data-intensiivistä sovellusta.

Tutustuin useisiin HTML5-pelimootteihin, joita käyttäisin pelin tekemisessä. Käytyäni läpi useita vaihtoehtoja päädyin Phaser.js-kirjastoon, jonka avulla sain peliin tuotua grafiikat, kontrollit ja muut pelilogiikkaan liittyvät asiat. Kirjaston mukana tuli myös fysiikan mallinnukseen käyttämäni yksinkertainen fysiikkamoottori.

Opinnäytetyöni toisessa luvussa tutustutaan projektissa keskeisessä osassa olevaan Node.js:ään ja siihen liittyviin laajennuksiin, joita olen sovelluksessani käyttänyt. Kolmannessa luvussa käydään läpi pelin tekemiseen ja julkaisuun liittyvä prosessi. Luku käsittelee muun muassa HTML5-pelimootteita, pelin sisältöön ja toteuttamiseen liittyviä asioita, reaaliaikaisen verkkopelin tuomia haasteita sekä verkkosivuston perustamiseen ja sitä kautta pelin julkaisemiseen liittyviä seikkoja. Neljännessä luvussa käsitellään opinnäytetyön prosessia kokonaisuudessaan ja pohditaan peliin liittyviä jatkokehitysmahdollisuuksia.



## 2 Node.js

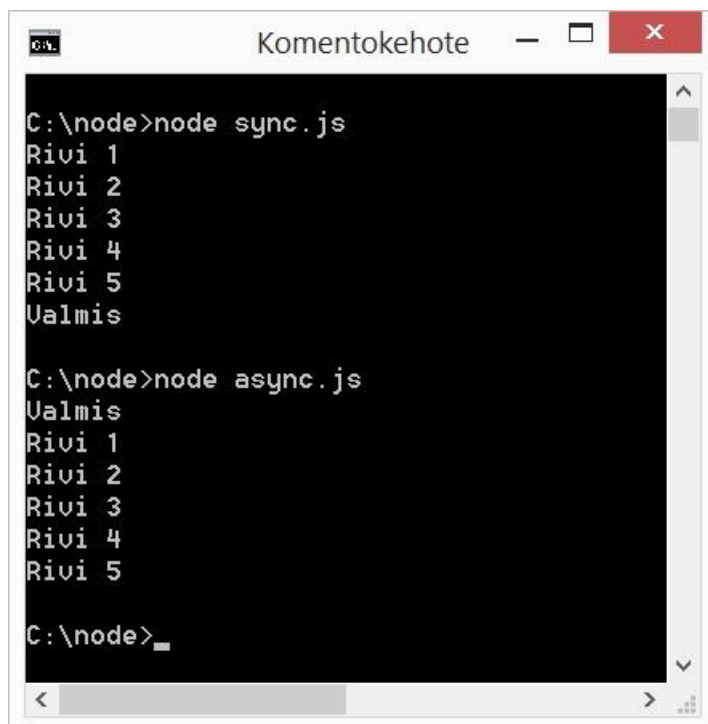
Node.js – jota yleisemmin kutsutaan yksinkertaisesti nimellä Node – on alusta, joka on rakennettu Google Chromen V8 -JavaScript-moottorin päälle. Node käyttää tapahtumapohjaista, ns. non-blocking I/O -mallia, jolloin asynkroninen toiminta saadaan palvelinpuolelle. Tämä tekee Nodesta kevyen ja tehokkaan työkalun data-intensiivisille reaaliaikaisille ohjelmistoille. (NodeJS 2014a.). Tämä on tärkeää monenlaisille reaaliaikaisille sovelluksille, kuten verkossa pelattaville nopeatempoisille moninpeleille.

Tarkastellaan yksinkertaista esimerkkiä kuinka asynkronisuus eroaa synkronisuudesta käytännössä. Kuvassa 1 on esitettynä yksinkertaisen tekstitiedoston lukeminen synkronisesti ja asynkronisesti.

SYNKRONINEN (BLOCKING)	ASYNKRONINEN (NON-BLOCKING)
<pre>var fs = require('fs');  var data = fs.readFileSync('teksti.txt','utf8'); console.log(data); console.log("Valmis");</pre>	<pre>var fs = require('fs');  fs.readFile('teksti.txt', 'utf8', function (er, data) {   console.log(data); }); console.log("Valmis");</pre>

Kuva 1. Tiedoston luku synkronisesti ja asynkronisesti.

Synkronisessa esimerkissä koodin suorittaminen pysähtyy tiedoston luvun ajaksi ennen kuin koodin ajaminen eteenpäin jatkuu. Asynkronisessa mallissa puolestaan koodin suorittaminen jatkuu eteenpäin ja tiedoston lukeminen tapahtuu taustalla. Kuvassa 2 näkyy konkreettisesti, kuinka asynkronisella menetelmällä koodinpätkässä viimeisenä oleva tulostus tapahtuu ennen tekstitiedoston sisällön tulostumista ja kuinka synkronisessa mallissa tiedoston sisältö tulostuu ensin.



```
C:\node>node sync.js
Rivi 1
Rivi 2
Rivi 3
Rivi 4
Rivi 5
Valmis

C:\node>node async.js
Valmis
Rivi 1
Rivi 2
Rivi 3
Rivi 4
Rivi 5

C:\node>
```

Kuva 2. Tulosteiden erilainen järjestys synkronisella ja asynkronisella tavalla.

Node käyttää JavaScript-ohjelmointikieltä palvelinpuolella. Tästä on paljon hyötyä verkkosovellusten kehittäjille: He pystyvät kirjoittamaan verkkosovelluksia yhdellä kielellä, mikä helpottaa asiakkaan ja palvelimen välistä yhteyden pitämistä. Täten esimerkiksi JavaScriptillä kirjoitetun pelin koodia pystyy käyttämään sellaisenaan sekä asiakas- että palvelinpuolella. JSON (JavaScript Object Notation) puolestaan on erittäin suosittu tiedonsiirtomuoto ja vaikka sitä pystyy käsittelemään myös muilla ohjelmointikielillä, se on alun perin johdettu JavaScript-kielestä, joten sitä on luonnollista ja helppoa käyttää siinä yhteydessä. JavaScript on myös kieli, joka toimii luonnostaan NoSQL-tietokantojen kanssa, jollainen tämänkin opinnäytetyön yhteydessä käytetty MongoDB on. (Cantelon, Harter, Holowaychuck & Rajlich 2013, 4–5.)

Node on kasvattanut suosiotaan ja sitä käytetään monessa korkean profiilin yrityksissä. Node on käytössä muun muassa Ebayn ql.io-projektissa, Paypalin verkkosovelluksissa, Yahoo!n Manhattan-projektissa, LinkedIn'in mobiilisovellusten palvelinrajapintana. Microsoft on pääasiallinen tukija Noden Windows-projektille ja tarjoaa Windows Azure -pilvipalvelun, jossa voi ajaa Node.js-sovelluksia. (NodeJS 2014b.)

## 2.1 Noden historia

Alun perin Noden kehittäjä Ryan Dahl sai inspiraation Noden kehittämiseen käyttäessään Flickr-palvelua, joka näytti tiedoston lähettämisen edistymisen ruudulla etenevänä palkkina. Selain ei kyennyt tietämään, kuinka paljon tiedoston lataaminen oli edistynyt, joten ohjelmiston täytyi tehdä jatkuvasti kyselyjä palvelimelle latauksen edistymisestä ja ilmoittaa siitä käyttäjälle. Tuohon aikaan tällainen monen pyynnön toteuttaminen yhdellä kertaa oli Dahlin mukaan hankalaa toteuttaa. Tämän ongelman johdosta idea Nodeen alun perin syntyi. (Harris 2014.)

V8 on C++-kielellä ohjelmoitu avoimen lähdekoodin JavaScript-moottori, joka on käytössä Google Chrome -selaimessa. V8 kääntää JavaScript-koodin konekielelle ennen sen ajamista sen sijaan, että se käyttäisi tulkkia välissä (Cantelon ym. 2013, 4). V8:ssa on myös tehokas automaattinen roskienkeruu, joka tarkoittaa sitä, että pyrkimyksenä on poistaa muistista automaattisesti tiedot, joita sovelluksessa ei enää tarvita. Tällöin vapautettu muistitila on uudelleen käytettävissä. Nämä edellä mainitut seikat ovat perustana V8:n tehokkuuteen. (Google Developers 2012.)

Dahlilla oli useita epäonnistuneita projekteja Noden suhteen hänen yrittäessään kehittää sitä C-, Lua- ja Haskell-ohjelmointikielillä. V8-JavaScript-moottorin ilmestyttyä Dahl huomasi, että JavaScript on täydellinen ohjelmointikieli, jonka avulla hän voisi saavuttaa tavoitteensa. Näin Dahl aloitti Noden työstämisen V8:aan perustuen. (McCarthy 2011.) Vuonna 2009 Dahl osallistui JSConf 2009 -tapahtumaan Berliinissä, jolloin hän esitteli projektiaan ensimmäisen kerran yleisölle ja sai heti hienon vastaanoton (W3resource 2014).

Node-projekti toimii Joyent-yhtiön johdolla. Ensimmäinen versio Nodesta julkaistiin Linux-käyttöjärjestelmälle vuonna 2011 ja myöhemmin samana vuonna Microsoft tuli tukemaan projektia, jolloin syntyi ensimmäinen versio Nodesta Windows-käyttöjärjestelmälle (NodeJS blog 2011a).

## 2.2 Npm

Npm on Noden paketinhallintajärjestelmä. Npm:n avulla Node-sovelluksiin pystytään asentamaan paketteja, toisin sanoen julkaistuja Node-sovelluksia. Myös itse tehtyjen sovellusten julkaisu onnistuu npm:n avulla. (Cantelon ym. 2013, 19.) Npm:n avulla käyttäjä voi itse asentaa vain tarvitsemansa laajennukset ja täten pitää sovelluksensa mahdollisimman pienenä ja kevyenä. Noden nykyversioissa npm tulee automaattisesti asennuksen mukana (NodeJS blog 2011b).

Nodessa muiden tekemien sovellusten asentaminen omaan sovellukseen on yksinkertaista konsolin kautta komennolla `"npm install [paketin nimi]"` (DreamersLab 2011). Jotta oma Node-sovellus olisi helposti käytettävissä muualla kuin perustetussa työkansiossa, sovelluksen riippuvuudet on hyvä laittaa package.json-tiedostoon. Package.json sijaitsee sovelluksen juurikansiossa ja on JSON-muotoinen tiedosto, jossa on määritelty sovellukseen liittyviä asioita kuten sovelluksen nimi, kuvaus, versionumero sekä riippuvuudet. (Cantelon ym. 2013, 19.)

Kuvassa 3 on esimerkki package.json-tiedostosta. Tärkeimmät kentät tiedostossa ovat ylimpänä olevat name- ja version-kentät, sillä ilman niitä pakettia ei edes pysty asentamaan. Dependencies-kohdassa on määritelty sovelluksessa tarvittavat riippuvuudet. Kun riippuvuudet on näin määritelty, ohjelma on helposti otettavissa käyttöön esimerkiksi toisella työasemalla tai toisessa työkansiossa. Package.json-tiedostossa määriteltyjen riippuvuuksien asentaminen käy yksinkertaisesti konsolin kautta komennolla `"npm install"`, jolloin kaikki listatut riippuvuudet asentuvat sen hetkiseen työkansioon, jos niitä ei siellä vielä asennettuna ole. Private-kenttä puolestaan määrittelee sen, onko kyseessä oleva sovellus mahdollista julkaista muille ladattavaksi, joten sen avulla pystytään esimerkiksi estämään sovelluksen julkaiseminen vahingossa. (NpmJS 2014.)

```
{
  "name": "Application name",
  "version": "0.0.1",
  "description": "My node application",
  "author": {
    "name": "My name",
    "email": "My email"
  },
  "homepage": "http://myappaddress/",
  "dependencies": {
    "express": "3.4.x",
    "socket.io": "0.9.x"
  },
  "private": true
}
```

Kuva 3. Esimerkki package.json-tiedostosta.

## 2.3 Projektissa käytetyt Node-paketit

### 2.3.1 Express

Express on pieni ja kevyt sovelluskehys, jonka avulla verkkosovelluksen kirjoittaminen Nodella on helppoa ja nopeaa. Expressin mukana tulee apuvälineet muun muassa tiedostojen lähettämiseen sekä URL:ien reitittämiseen. Express tarjoaa myös systeemin, jonka kanssa toimivat monet erilaiset template-enginet, kuten tämän opinnäytetyön yhteydessä käytetty Jade. (Cantelon ym. 2013, 176.) Template-enginen ansiosta verkkosivujen kirjoittaminen on yksinkertaista, kunhan ensin opettelee enginen syntaksin. Kuvassa 4 on esimerkki, kuinka yksinkertainen HTML-sivu kirjoitetaan Jaden avulla. Kuten esimerkissä näkyy, Jadella ei tarvitse lainkaan kirjoittaa avaus- ja sulkutageja, vaan avaukset ja sulut muodostuvat rivien sisennysten mukaan.

HTML:	Jade:
<pre>&lt;!doctype html&gt; &lt;html&gt;   Welcome!   &lt;head&gt;     &lt;title&gt;Hello&lt;/title&gt;   &lt;/head&gt;   &lt;body&gt;     &lt;h1&gt;Hello world!&lt;/h1&gt;   &lt;/body&gt; &lt;/html&gt;</pre>	<pre>doctype html html     Welcome!   head     title Hello   body     h1 Hello world!</pre>

Kuva 4. Yksinkertainen esimerkki HTML:n kirjoittamisesta Jadella.

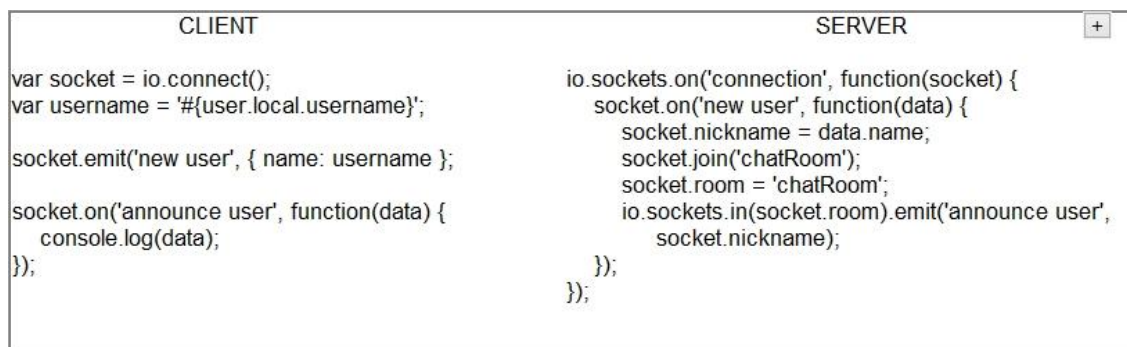
### 2.3.2 Socket.io

Socket.io on luultavasti parhaiten tunnettu paketti Node-yhteisössä. Socket.io:n avulla onnistuu reaaliaikaisen kaksisuuntaista kommunikointikanavaa palvelimen ja asiakkaan välillä käyttävän web-sovelluksen kehittäminen. Socket.io:n mukana tulevien ohjelmointirajapintojen avulla siitä on tullut erittäin suosittu esimerkiksi selainpeleissä ja chat-sovelluksissa. (Cantelon ym. 2013, 310.)

HTTP on ns. tilaton protokolla, joka tarkoittaa sitä, että asiakas pystyy tekemään yksittäisiä lyhytikäisiä pyyntöjä palvelimelle. WebSocket kuitenkin tuo selainten ja palvelinten välille jatkuvan kaksisuuntaisen yhteyden, jolloin molemmat päät pystyvät lähettämään ja vastaanottamaan tietoa samanaikaisesti. (Cantelon ym. 2013, 310.)

Socket.io käyttää pääsääntöisesti WebSocket-protokollaa. Jos käytettävässä olevassa selaimessa ei löydy tukea WebSocketille, Socket.io osaa vaihtaa mukana tuleville toisille tekniikoille simuloiden WebSocketin toimintaa. Tällöin sovellus toimii myös selaimissa, joissa WebSocket tukea ei ole. (Cantelon ym. 2013, 310.)

Kuvassa 5 on esitetty yksinkertainen esimerkki Socket.io:n toiminnasta. Jokaisella liittyneellä asiakkaalla on oma socketinsa, johon voi määritellä asiakkaaseen liittyviä asioita. Esimerkissä näkyy, kuinka uuden asiakkaan liittyessä asiakkaan päästä lähetetään viesti 'new user'. Kun palvelin saa vastaan tämän viestin, palvelinpäässä määritellään asiakaskohtaisia tietoja ja hänet lisätään huoneeseen, joka on nimeltään chatRoom. Tämän jälkeen palvelin lähettää viestin 'announce user' vain niille asiakkaille, jotka ovat liittyneenä samaan huoneeseen. Täten viestien vastaanottajien laajuutta voidaan rajoittaa ja välittää viestin lähettämistä asiakkaille, joille sitä ei ole tarkoitettu. Näin esimerkiksi peliä tehtäessä voidaan jokaiselle peli-instanssille määritellä oma huoneensa, jolloin asiakkaan lähettämät tiedot pelin aikana lähetetään ainoastaan samassa huoneessa eli samassa pelissä mukana oleville asiakkaille.



Kuva 5. Esimerkki Socket.io:n toiminnasta.

### 2.3.3 Passport

Passport.js on Nodelle tehty väliohjelmisto, jonka avulla Express-pohjaiseen Node-sovellukseen saadaan helposti lisättyä käyttäjien rekisteröityminen ja kirjautuminen. Passportin avulla tunnistautumisen pystyy tekemään usealla eri tavalla. Perinteisen tyylin, jossa kirjaudutaan käyttäjänimen ja salasanan avulla, lisäksi Passport tarjoaa myös OAuth-tunnistautumisen erilaisten palvelujen kuten Google-, Facebook- ja Twitter-tilien avulla. (Passport 2013.)

Kun Passportin asetukset on laitettu kohdilleen, sen avulla kirjautuminen ja rekisteröityminen hoituu helposti vähäisellä koodin määrällä. Tässä projektissa näin tarpeelliseksi tehdä ainoastaan perinteisen rekisteröitymis- ja kirjautumissysteemin, jolloin käyttäjänimeä käytetään samalla ajajan nimenä. Rekisteröityneiden käyttäjien kirjautumistiedot tallennetaan MongoDB-tietokantaan. Tietokannassa sijaitsevien salasanojen salaamiseen käytin bcrypt-nodejs-kirjastoa.

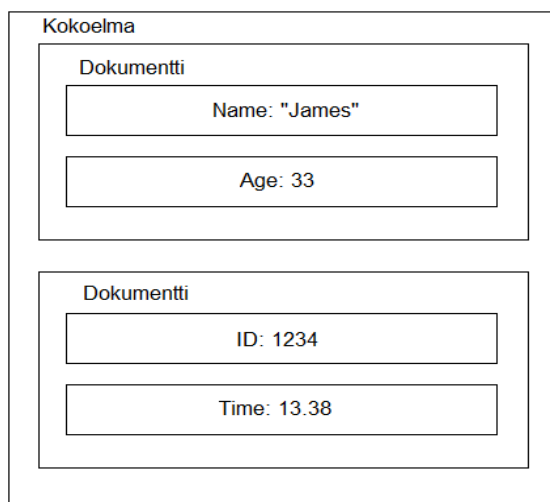
Passportin avulla saa helposti myös tehtyä rajoituksia sivuston eri osioille, jolloin käyttäjä ei pääse kirjautumatta katsomaan suojattuja alueita. Tällöin jos käyttäjä ei ole tunnistautunut ja pyrkii pääsemään suojatulle alueelle sivustolla, hänet voidaan esimerkiksi ohjata kirjautumissivulle.

### 2.3.4 Mongoose

Tietokantojen alkuaikoina tietokannat olivat yleisesti relaatiomallista poikkeavia. Relaatiotietokannat kuitenkin yleistyivät pikkuhiljaa ja ajan myötä niitä käytettiin yleisesti kaikenlaisissa sovelluksissa. Viime vuosina relaatiomallista poikkeavat tietokantasysteemit ovat kuitenkin jälleen yleistyneet niiden kannattajien esittäessä etuja skaalautuvuudessa ja yksinkertaisuudessa. Yleisesti tällaisia relaatiotietokannoista poikkeavia tietokantoja kutsutaan NoSQL-tietokannoiksi. ”NoSQL” voidaan tulkita suoraan ”No SQL” (ei SQL) lisäksi ”Not Only SQL” (ei ainoastaan SQL). (Cantelon ym. 2013, 112.)

Relaatiotietokannoissa suorituskky kärsii luotettavuuden kustannuksella, kun taas NoSQL:n kohdalla suorituskky on laitettu etusijalle. NoSQL-tietokannat eivät myöskään tavallisesti tarvitse minkäänlaista kiinteää rakennetta kuten relaatiotietokannat. (Cantelon ym. 2013, 112.)

MongoDB on yksi suosituimmista NoSQL-tietokannoista. Sana MongoDB muodostuu englannin kielen sanasta **humongous**, joka suomeksi tarkoittaa valtavaa (MongoDB 2013). MongoDB-tietokanta koostuu kokoelmista, jotka puolestaan koostuvat dokumenteista. NoSQL-tietokannalle tyypillisesti näiden dokumenttien ei tarvitse keskenään olla rakenteeltaan samanlaisia. Kuvassa 6 on esitetty esimerkki, kuinka kokelmassa sijaitsevat dokumentit voivat olla rakenteeltaan erilaisia.



Kuva 6. Dokumenttien rakenne MongoDB-tietokannassa voi vaihdella.



Mongoose on Node-sovellus, joka tekee MongoDB:n käyttämisestä Node-sovelluksessa helppoa. Mongoose tarjoaa suoraviivaisen, skeemapohjaisen ratkaisun sovelluksen datan mallintamiseen. (MongooseJS 2014.)

Mongoosessa käytetään skeemoja, jotka määrittelevät kokoelmassa sijaitsevien dokumenttien muodon (MongooseJS guide 2014). Seuraavassa koodinpätkässä on esimerkki tällaisen skeeman määrittämisestä.

```
var Schema = mongoose.Schema;

var blogSchema = new Schema({
  title: String,
  author: String,
  body: String,
  date: { type: Date, default: Date.now }
});
```

### 3 HTML5-pelin kehittäminen

Opinnäytetyössä toteutetun pelin idea perustuu suomalaisen Timo Kauppisen kehittämään Slicks 'n Slide -tietokonepeliin. Slicks 'n Sliden ensimmäinen versio julkaistiin PC:lle DOS-käyttöjärjestelmälle vuonna 1993. Pelin ideana on perinteinen autokilpailu, eli tarkoituksena on kiertää ajoneuvolla rataa ennaltamäärätyn kierrosmäärän verran ja tavoitteena on päästä ensimmäisenä maaliin. Slicks 'n Slidessa on monia erilaisia ajoneuvotyyppiejä, joissa ominaisuudet kuten nopeus ja ohjautuvuus vaihtelevat. Pelissä pystyy ajamaan kisoja sekä tekoälyvastustajia että samalla tietokoneella pelaavia muita ihmispelaajia vastaan.

Tässä opinnäytetyössä tehtävään peliin oli tavoitteena saada samanlainen tunta ja hauskuuden tunne kuin esikuvassa Slicks 'n Slidessa. Suurimpina eroina esikuvaansa verrattuna peli kehitettiin selaimelle ja siihen lisättiin moninpeleimahdollisuus verkossa, jolloin pelaajat voivat ajaa toisiaan vastaan omilla tietokoneillaan verkon välityksellä. Tässä prototyyppivaiheessa peliin oli tavoitteena saada vain toimiva pelirunko, eikä peliin ollut tarvetta vielä saada mukaan

esimerkiksi erilaisia ajoneuvoja, tekoälyä eikä aseistuksia. Nämä voivat olla tulevaisuudessa mahdollisessa jatkokehityksessä mukana.



Kuva 7. Kuvankaappaus Slicks 'n Slide –pelistä (Slicks 'n Slide 1993).

### 3.1 HTML5-pelimoottorit

Verkosta on ladattavissa useita erilaisia pelimoottoreita HTML5-pelin tekemiseen. Ladattavia vaihtoehtoja on lukuisia, joista osa on maksullisia, mutta suuri osa pelimoottoreista on kuitenkin ilmaisia avoimen lähdekoodin kirjastoja. Useat ilmaiset HTML5-pelimoottorit ovat ominaisuuksiltaan ainakin lähestulkoon tasavertaisia maksullisten moottoreiden kanssa.

Halusin käyttää projektissani ilmaisia työkaluja, ja asiaan perehdyttyäni päädyin Phaser-pelimoottoriin ja sen versioon 1.1.6. Valitsin käyttööni Phaserin, koska sen mukana tuli muun muassa kevyt fysiikan mallinnus ja WebGL-tuki. Valintaa tuki myös se, että internetistä löytyy Phaserilla tehtyjä hyviä koodiesimerkkejä ja Phaserin dokumentointi on hyvällä tasolla. Phaseria myös kehitetään aktiivisesti ja tätä kirjoittaessa se onkin jo edennyt versioon 2 sisältäen muun muassa vaihtoehtoiset monipuolisemmat fysiikanmallinnuskirjastot integroituna.

### 3.1.1 Phaser

Phaser.js on avoimen lähdekoodin kirjasto, jonka avulla pystyy tekemään HTML5-pelejä tietokoneiden ja mobiililaitteiden verkkoselaimille. Kirjaston ominaisuuksiin kuuluu muun muassa fysiikanmallinnus, animaatiot, partikkelit, kameratoiminnot, äänet ja TileMappien käyttö. Renderöinnissä Phaser käyttää Pixi.js-kirjastoa, joka tukee sekä Canvas- että WebGL-renderöintiä. Phaser osaa tarvittaessa vaihtaa renderöintiä niiden välillä riippuen siitä, minkälainen tuki käytössä olevassa selaimessa on. (Phaser 2014.) WebGL on järjestelmäriippumaton ohjelmointirajapinta, jonka avulla verkkoselaimessa saadaan toteutettua laitteistokiihdytettyä 3D-grafiikkaa (WebGL wiki 2011). Vaikka tämän projektin yhteydessä tehty autopeli onkin kaksiulotteinen, on WebGL:stä hyötyä sen tarjoaman laitteistokiihdytyksen ansiosta.

Kuvassa 8 on esitettyä Phaser-sovelluksen perusrunko. Riveillä 1 ja 2 luodaan instanssi Phaser.Game-luokasta, joka saa useita parametreja. Ensimmäiset kaksi parametria määrittelevät pelialueen koon, tässä tapauksessa 800 x 600 pikseliä. Toinen parametri määrittelee käytettävän renderöijän, ja se voi olla Phaser.CANVAS, Phaser.WEBGL, Phaser.AUTO tai Phaser.HEADLESS, jolloin renderöintiä ei tehdä lainkaan. Phaser.AUTO yrittää ensin käyttää WebGL-renderöintiä, mutta jos selain tai laite ei tue sitä, renderöinti siirtyy automaattisesti canvasille. Kolmas parametri on DOM-elementin tunniste, johon Phaserin luoman canvas-elementin haluaa laitettavaksi. Kohdan voi jättää tyhjäksi, jolloin canvas muodostuu automaattisesti verkkosivun runkoon. (Photon Storm 2013.)

```

var game = new Phaser.Game(800, 600, Phaser.AUTO, "",
    { preload: preload, create: create, update: update, render: render });

function preload() {
    game.load.image('car', 'assets/car.png');
}

var car;
var checkpoint;

function create() {
    car = game.add.sprite(100, 50, 'car');
    checkpoint = new Phaser.Rectangle(50, 0, 20, 100);
}

function update() {
    getInput();
}

function render() {
    game.debug.renderRectangle(checkpoint, "white");
}

```

Kuva 8. Phaser-sovelluksen perusrunko.

Phaser-sovelluksessa preload-funktio on tarkoitettu pelissä käytettävien assettien lataamiseen. Create-funktio ajetaan sen jälkeen, kun preload-funktio on käsitelty, ja on tarkoitettu peliobjektien lisäämiseen ja niiden ominaisuuksien säätämiseen. Lisättäessä renderöitäviä objekteja täytyy muistaa, että järjestys, jossa objektit renderöityvät, määräytyy sen mukaan, missä järjestyksessä ne on luotu. Toisin sanoen myöhemmin lisätyt objektit renderöityvät aiemmin lisättyjen objektien päälle. Kun create-funktio on suoritettu, pelin pääsilmutka käynnistyy. Silmutka kutsuu update- ja render-funktioita jokaisella framella, joka tapahtuu tavallisesti 60 kertaa sekunnissa. Tämä tarkoittaa, että update- ja render-funktioihin kannattaa laittaa vain tarvittavat toiminnallisuudet, jotta turhaan käsiteltävät asiat eivät vähentäisi pelin suorituskykyä. Render-funktiota kutsutaan update-funktion ja WebGL/canvas-renderöinnin jälkeen. Render-funktio on kätevä debuggauksessa, jolloin on helppo tarkistaa, onko lisätty näkymätön objekti sijoittunut oikealle paikalle, kuten esimerkissä oleva tarkistuspiste. Debuggauksen renderöinnit eivät näy lainkaan WebGL-renderöinnillä, eli ainakin debuggauksen ajaksi on renderöinti vaihdettava canvas-renderöinnille. (Phaser General Documentation 2014a.)

Phaser-pelissä on vähintään yksi state eli pelitila. Tiloja pystyy lisäämään esimerkiksi peliobjektien lataamista varten, pelin alkuvalikolle ja käynnissä olevalle pelille. Jokaisella tilalla on käytössä varatut funktiot, joita ovat aiemmin mainittujen preload-, create-, update- ja render-funktioiden lisäksi loadUpdate-funktio, jota kutsutaan preload-funktion ollessa käsittelyssä ja on tarkoitettu peliobjektien lataamisen edistymisen esittämiseen. LoadRender-funktiota kutsutaan heti loadUpdate-funktion jälkeen, paused-funktiota kutsutaan kun peli on paussilla ja shutdown-funktiota kutsutaan, kun tila-objekti tuhotaan. Nämä edellä mainitut funktiot ovat siis varattuja funktioita eivätkä ne tee mitään, ennen kuin ohjelmoija kirjoittaa niiden yli jotain. Jotta luotua tilaa pystyy käyttämään, täytyy vähintään yksi varatuista funktioista ylikirjoittaa. (Phaser General Documentation 2014a.)

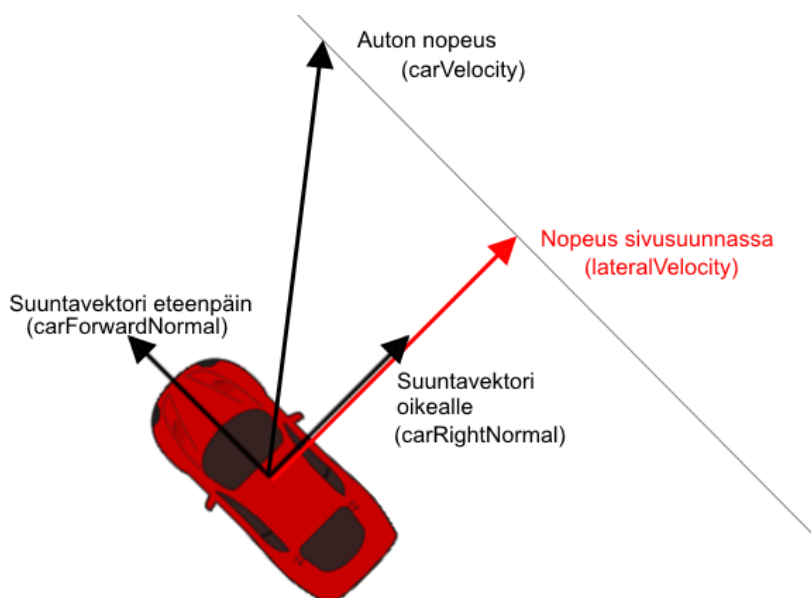
Tilojen välillä pystyy siirtymään ja sen mukaan missä tilassa ollaan, varattuja funktioita ja tietenkin mahdollisia omia funktioita ajetaan. Phaserin StateManager-luokka on vastuussa tilojen käsittelystä. Esimerkiksi kun tilaa vaihdetaan, StateManager pyyhkii kaiken, kuten peliobjektit, tilasta josta poistutaan. (Phaser General Documentation 2014b.)

### 3.2 Fysiikan mallintaminen

Koska toteuttamani peli on arcade-tyyppinen autopeli, lähestyin auton ajamiseen liittyvää fysiikan mallintamista siten, että auton liikkumisen ei tarvitse eikä pidäkään olla realistista. Pääasia tällaisessa pelissä kuitenkin on, että autolla ajaminen on hauskaa ja koukuttavaa.

Tässä projektissa oli käytössä Phaserin versio 1.1.6. Tämän version mukana tulee Arcade Physics -fysiikkamoottori, joka on pieni ja kevyt AABB-kirjasto (Axis Aligned Bounding Box) ja jonka avulla peliin saa helposti perustoiminnot fysiikan mallintamiseen, kuten esimerkiksi auton kontrollointiin ja törmäyksen tunnistamiseen.

Auton kääntymisen tein samalla tavalla kuin esikuvassa Slicks 'n Slidessa eli siten, että autoa pystyy kääntämään paikallaan ilman vauhtia. Kiihdyttäminen ja jarruttaminen oli myös yksinkertaista asettaa Phaserin tarjoamien ominaisuuksien avulla. Perehtyessäni auton käyttäytymiseen mutkissa totesin, että käyttäytymisen saaminen tuntumaltaan hyväksi vaatii hieman enemmän työtä. Auton saa helposti kulkemaan asettamalla auton nopeuden suunnan auton rotaation mukaan, mutta tällöin auto kulkee kuin kiskoilla. Pelattavuuden kannalta tämä ratkaisu ei ole toimiva, joten auton käyttäytymiseen kääntymisessä täytyi löytää parempi ratkaisu. Tarkastellaan ongelman ratkaisua kuvassa 9 näkyvien tietojen avulla, jossa auto on sivuttaisluisussa, eli auton nopeuden suunta ei ole keulan osoittamassa suunnassa.



Kuva 9. Auton vektoreita.

Auton sivuttaisnopeuden selvittämiseksi täytyy ensin ratkaista auton normaali-vektorit eteenpäin ja sivusuuntaan. Käytössä ollut Phaserin versio ei sisällä valmiita ominaisuuksia näihin, mutta nämä vektorit voi määritellä auton rotaatiota käyttäen seuraavilla kaavoilla:

### Suuntavektori eteenpäin

```
carForwardNormal = new Phaser.Point(
    Math.cos(carRotation*Math.PI/180),
    Math.sin(carRotation*Math.PI/180));
```

**Suuntavektori oikealle**

```
carRightNormal = new Phaser.Point(
    Math.cos((carRotation+90)*Math.PI/180),
    Math.sin((carRotation+90)*Math.PI/180));
```

Kun nämä vektorit ovat tiedossa, nopeudet eteenpäin ja sivusuuntaan auton rungon suuntaan nähden saadaan kertomalla normaalivektorin ja auton nopeuden pistetulo edelleen normaalivektorilla seuraavalla tavalla:

**Nopeus eteenpäin**

```
fwdVelocity =
    carForwardNormal * dotProduct(carVelocity, carForwardNormal);
```

**Nopeus sivusuuntaan**

```
lateralVelocity =
    carRightNormal * dotProduct(carVelocity, carRightNormal);
```

Nyt kun nopeudet eteenpäin ja sivusuuntaan ovat tiedossa, saadaan vähennettyä sivusuunnassa olevaa nopeutta haluttu määrä seuraavasti:

**Auton nopeus x-akselilla**

```
carVelocity.x = fwdVelocity.x + lateralVelocity.x*drift;
```

**Auton nopeus y-akselilla**

```
carVelocity.y = fwdVelocity.y + lateralVelocity.y*drift;
```

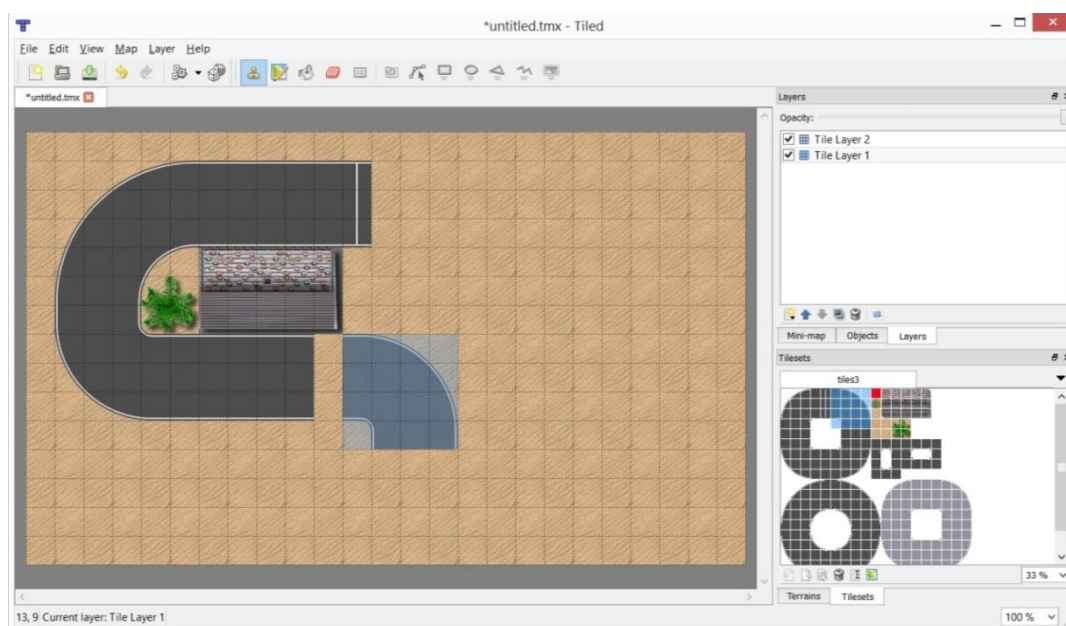
Täten muuttujan drift arvoa muuttamalla saa muutettua sivusuunnassa olevan nopeuden määrää. Muuttujan drift arvo tulee olla lukujen 0 ja 1 välissä, eli arvon ollessa 1 auton sivuttaisnopeutta ei vähennetä lainkaan, jolloin auton kulkusuunta ei muutu mitenkään, jos autoa käännetään ilman kaasua, eli auton käyttäytyminen on verrattavissa avaruusaluksen liikkumiseen avaruudessa. Arvolla 0 kaikki sivuttaissuuntaan tapahtuvat nopeudet poistetaan, toisin sanoen auto kulkee aina keulan osoittamaan suuntaan aivan kuin kiskoja pitkin. Täten muuttujan arvoa vaihtelemalla saa auton käyttäytymistä muutettua esimerkiksi

silloin, kun auto kulkee jäisellä tiellä. Käytännössä drift-muuttujan arvoa tulee käytettyä lähellä yhtä, koska lauseketta suoritettaessa jokaisella framella, auton sivuttaisnopeus putoaa liian nopeasti muuttujan arvon ollessa liian pieni.

### 3.3 Ratojen suunnittelu

Phaser sisältää ominaisuudet TileMappien käsittelyä varten. Phaser tukee TileMapeissa CSV- ja JSON-formaatteja. (Phaser 2014.) Käytin projektissani ilmaista Tiled-karttaeditoria, jonka avulla suunnittelin radan ja talletin sen JSON-muodossa, jonka sai helposti lisättyä peliin.

TileMap tarkoittaa nimensä mukaisesti tiilistä (ruuduista) koostuvaa aluetta. TileSet puolestaan on kuvatiedosto, joka on jaoteltu ruudukoksi. Näitä ruutuja eli tiilejä käytetään rakennuspalikoina pelialuetta rakennettaessa. TileMap voi koostua useasta tasosta (layer), jolloin jokaiselle tasolle pystyy erikseen määrittelemään esimerkiksi törmäyksiin liittyviä toimintoja. Suunnittelutyössä kannattaa kuitenkin muistaa, että suuri määrä tasoja voi vaikuttaa negatiivisesti suoritussykyyn erityisesti tällaisessa JavaScriptillä kirjoitetussa HTML5-pelissä. Myös TileMapin tiilien kokoa ei kannata asettaa liian pieneksi, jotta käsiteltäviä tiiliä ei tulisi liikaa, jolloin ne vähentävät suoritussykyä.



Kuva 10. Radan suunnittelua Tiled-karttaeditorilla.



Phaser tarjoaa mahdollisuuden laittaa törmäyksen tunnistus tiilille. Tiilille voi asettaa collision-ominaisuuden, jolloin auto ei pysty ajamaan niistä läpi vaan törmää niihin. Tiilille on myös mahdollisuus asettaa callback-funktioita, jolloin auton ollessa tällaisen tiilen kanssa kosketuksessa suoritetaan jokin määritelty funktio. Esimerkiksi kuvassa 10 näkyville hiekkaa esittävälle tiilille pystyy asettamaan callbackin siten, että auton osuessa niihin ajetaan funktio, joka vähentää auton nopeutta.

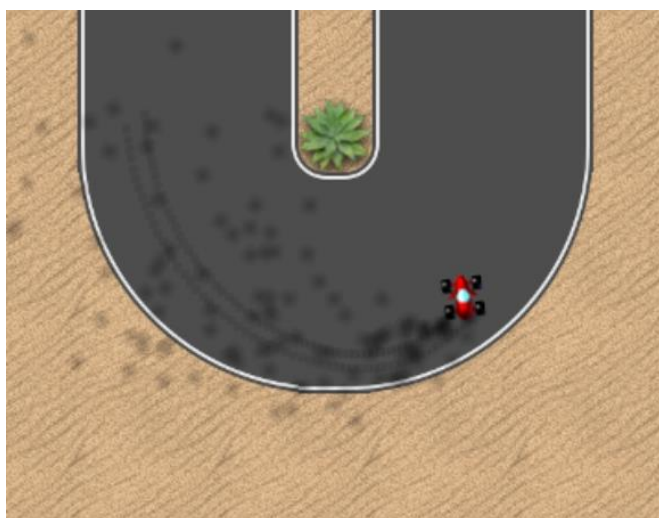
Jotta pelaajat eivät pysty oikaisemaan radalla sellaisissa paikoissa, joissa siitä olisi ajallisesti hyötyä, lisäsin rataa checkpointit eli tarkistuspisteet. Tarkistuspisteet ovat yksinkertaisesti lista radalle lisätyistä suorakulmioista Phaserin Rectangle-luokan avulla. Auton osuessa tällaiseen tarkistuspisteeseen asetetaan tarkistuspisteellä vierailun arvoksi tosi edellyttäen, että pelaaja on käynyt myös edellisellä tarkistuspisteellä. Maalilinjalle on määritelty oma tarkistuspisteensä, johon osuttaessa suoritetaan tarkistus käydyistä tarkistuspisteistä. Jos pelaaja on maalilinjan ylittäessään vieraillut kaikilla tarkistuspisteillä, kierros hyväksytään. Suorakulmioiden sijaintien määrittämisessä hyvänä apuna toimi Phaserin render-funktio, jonka avulla suorakulmiot saa näkyville pelialueelle, kun renderöijäksi asettaa canvasin.

### 3.4 Grafiikat

Pelissä käytettävät graafiset elementit pyrin hakemaan internetistä siten, että niitä ei kaikkia tarvitsisi itse piirtää. Käytin hyödykseni OpenGameArt.org-sivustoa, josta voi hakea vapaaseen käyttöön tarkoitettuja grafiikoita ja ääniä. Osa pelissä käytetyistä grafiikoista, kuten katsomoihin, puihin ja autoihin liittyvät grafiikat, ovat peräisin Turbo Sliders -pelistä, joka on Antti Manniston tekemä ja Jollygood Gamesin markkinoima ylhäältäpäin kuvattu ajopeli. Turbo Sliders on eräänlainen Slicks 'n Slide -klooni ja siinä on jopa 20 pelaajan verkkopelituki. Turbo Slidersista lainaamani grafiikat ovat vapaasti käytettävissä GPL-lisenssin alaisena.

Pelin näytävyyttä parantaakseni halusin saada siihen mukaan auton renkaiden jättämät jäljet sekä savu- ja jääefektit. Renkaiden jälkien lisääminen onnistui Phaserin `RenderTexture`-luokan avulla, joka on erikoistekstuuri, ikään kuin tyhjä canvas-elementti, johon voi renderöidä objekteja (Phaser Docs 2014a). Täten peliin lisätyn kuvatiedoston, joka tässä tapauksessa esittää yhtä pientä renkaan jättämää jälkeä, saa piirrettyä silloin, kun autolla on esimerkiksi tarpeeksi sivuttaissuuntaista nopeutta. Kun `RenderTexture`-luokasta luotu tekstuuuri on määritetty pelialueen kokoiseksi, renkaan jäljet saa piirrettyä pelialueelle laskemalla auton renkaiden positiot ja piirtämällä objektit näihin pisteisiin. Koska pelin prototyypissä ovat mukana myös jäiset tieosuudet, asetin renkaiden jäljille erilaiset grafiikkaobjektit jäisille osuuksille. Tällöin luvussa 3.3 esitetyn `TileMap`in callback-funktion avulla auton ollessa jäisellä alueella, näytölle piirretään erilaiset renkaiden jäljet.

Savu- ja jääefektit puolestaan pystyi tekemään Phaserin tarjoamalla `Emitter`-luokalla. `Emitter`-luokan avulla onnistuu partikkelien luominen lyhytkestoisissa tapahtumissa, kuten räjähdyksissä, tai jatkuvissa efekteissä, kuten sateen tai tulen luomisessa. Phaserissa partikkeleille voi asettaa fyysisiä ominaisuuksia, kuten maan vetovoiman vaikuttamaan niihin, tai asettaa ne törmäämään muihin peliobjekteihin. (Phaser Docs 2014b.) Näissä savu- ja jääefekteissä tällaisia fyysisiä ominaisuuksia ei kuitenkaan tarvita, vaan efektit on hyvä vain renderöidä lisätystä kuvaobjektista ruudulle silloin, kun renkaat jättävät jälkiä ajoalustalle, ja asettaa ne häviämään näkyvistä määritellyn ajan kuluttua.



Kuva 11. Renkaiden jäljet ja savuefektit.



Kuva 12. Efektit jäisellä tieosuudella.

Grafiikoihin, joita käsittelin ja tein, käytin GIMP-sovellusta (GNU Image Manipulation Program). GIMP on monipuolinen ja ilmainen sovellus grafiikan tuottamiseen ja muokkaamiseen ja valokuvien käsittelyyn. (GIMP 2013.)

## 3.5 Äänet

### 3.5.1 HTML5 Audio

Ennen HTML5:tä verkkosivuilla ei voinut toistaa ääniä ilman erikseen asennettavia lisäosia, kuten Flashia tai QuickTimea. HTML5:n ominaisuuksiin kuuluu audio-elementti, jonka avulla verkkosivulla pystyy toistamaan ääniä. Tämä elementti tukee MP3-, Wav- ja Ogg-tiedostomuotoja. Kuvassa 13 on esitetty, kuinka eri selaimet tukevat näitä tiedostoformaatteja. (W3schools 2014.)

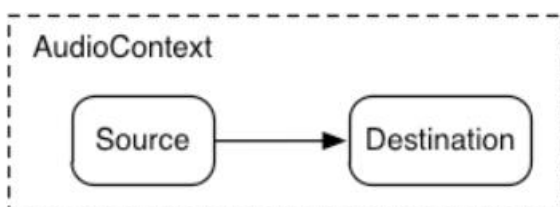
Browser	MP3	Wav	Ogg
Internet Explorer	YES	NO	NO
Chrome	YES	YES	YES
Firefox	NO <b>Update:</b> Firefox 21 running on Windows 7, Windows 8, Windows Vista, and Android now supports MP3	YES	YES
Safari	YES	YES	NO
Opera	NO	YES	YES

Kuva 13. Selaimien tuki HTML5 Audion tiedostoformaateille (W3schools 2014).

HTML5 Audio saattaa riittää yksinkertaisissa toteutuksissa, mutta se on melko rajoittunut eikä usein ole tarpeeksi tehokas monimutkaiseen äänen tuottamiseen. Esimerkiksi pelit tarvitsevat usein monipuolisemman ratkaisun, ja tällaisia tilanteita varten on kehitetty Web Audio -ohjelmointirajapinta. (Rogers 2013.)

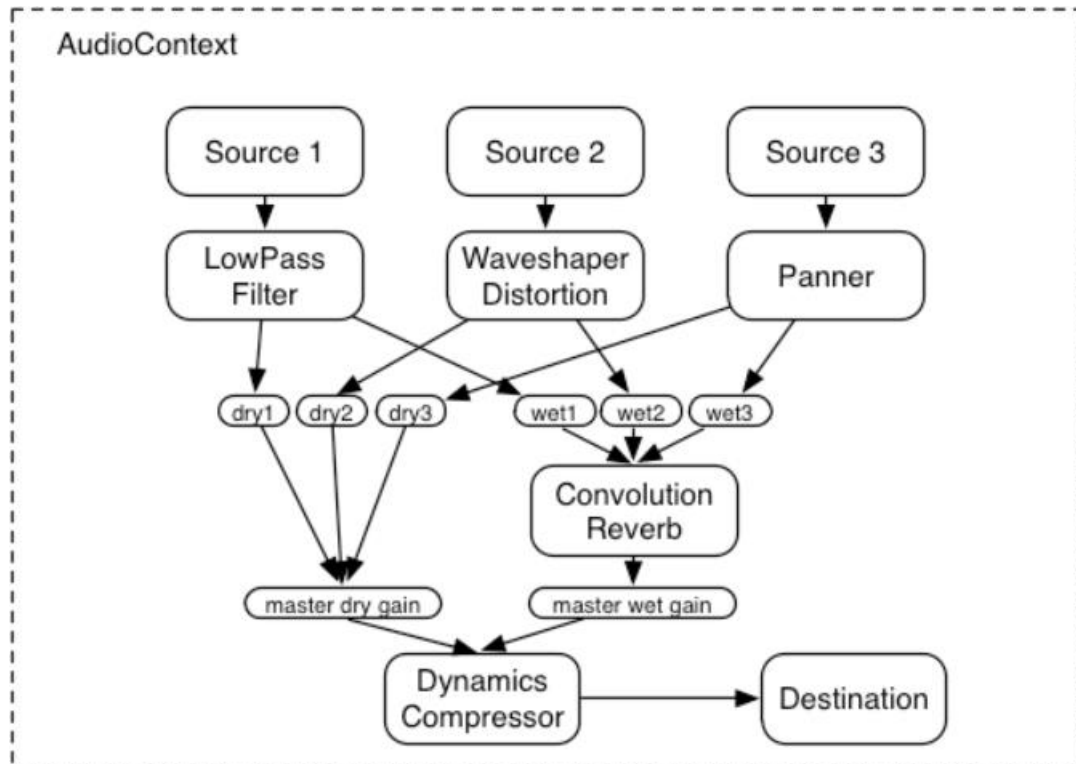
### 3.5.2 Web Audio

Web Audio on JavaScript-ohjelmointirajapinta äänen tuottamiseen verkkosovelluksissa. Web Audiossa äänien muodostaminen tapahtuu luomalla AudioContext-instanssi, joka käsittelee ja toistaa äänet. AudioContext koostuu AudioNodeista, jotka voivat olla esimerkiksi äänilähteitä, erilaisia efektejä tai loppukohde, jossa äänet toistetaan. Äänilähteitä voi olla useita ja ne voivat käydä erilaisen reitin ennen kuin ne yhdistetään loppukohteessa toistettavaksi. Kuvassa 14 on esitetty Web Audion toimintamalli yksinkertaisimmillaan, jossa yksi äänilähde (source) on ohjattu suoraan toistettavaksi. (Rogers 2013.)



Kuva 14. Yksinkertainen esimerkki Web Audion toiminnasta (Rogers 2013).

Kuvassa 15 on esitetty monimutkaisempi malli, jossa mukana on kolme äänilähdettä, jotka käyvät läpi erilaisen reitin esimerkiksi erilaisten lisäefektien kautta ennen yhdistymistä ja toistumista muiden äänilähteiden kanssa.



Kuva 15. Monimutkaisempi esimerkki Web Audion toiminnasta (Rogers 2013).

Phaserin mukana tulee tuki HTML5 Audiolle ja Web Audio-ohjelmointirajapinnalle. Phaser sisältää SoundManager-luokan, jonka avulla äänet saadaan helposti otettua käyttöön Phaser-sovelluksessa, jos selain näitä ominaisuuksia tukee. (Phaser Docs 2014c.)

### 3.6 Verkkopelin kehittämisen haasteet

Videopelin tekeminen on haastavaa. Kun peliin lisätään moninpelimahdollisuus verkossa, pelin tekemisestä tulee vielä paljon haastavampaa. Tässä luvussa tarkastellaan reaaliaikaisen moninpelin tekemiseen liittyviä haasteita ja haetaan niihin ratkaisuja.

### 3.6.1 Huijaaminen

Moninpelissä on aina vaarana, että pelaajat huijaavat. Eräs keino, jolla huijaamista voidaan hankaloittaa, on käsittelemällä kaikki pelin tapahtumat palvelimella. Toisin sanoen pelaaja lähettää palvelimelle ainoastaan syötteitä, kuten näppäinpaineluja. Syötteet käsitellään palvelimella ja niiden aiheuttamat tulokset lähetetään pelaajille. (Gambietta 2014.)

Palvelin ei tietenkään ole haavoittumaton, mutta kun pelin tapahtumat käsitellään palvelimella, estetään monenlaisia huijausmahdollisuuksia. Esimerkiksi jos pelaajien tietoja käsitellään paikallisesti heidän omilla koneillaan, pelaajalle avautuu mahdollisuus muuttaa näitä tietoja. Näin osaava pelaaja voi vaihtaa esimerkiksi omaa positiota pelialueella, muuttaa auton tehoja suuremmiksi tai vaikka asettaa pelihahmoon aiheutuneet vauriot nollaan. Kun nämä muutetut tiedot lähetetään palvelimelle, se käsittelee tapahtumat näiden tietojen perusteella. (Gambietta 2014.)

### 3.6.2 Latenssi

Latenssilla tarkoitetaan sitä aikaa, mikä verkon kautta kulkevalta paketilta kuluu lähettäjältä vastaanottajalle. Edellisessä luvussa esitelty malli, jossa palvelin käsittelee kaiken tiedon, toimii erinomaisesti hitaassa vuoropohjaisessa moninpelissä, kuten shakissa tai korttipelissä, jossa tapahtumien ei tarvitse näkyä pelaajalle nopeasti ilman viivettä. Malli voi toimia hyvin myös nopeatempoisissa reaaliaikaisissa peleissä, jos peliä pelataan paikallisverkossa, jossa latenssit ovat tyypillisesti hyvin alhaisia. Jos nopeatempoista ja reaaliaikaista peliä pelataan hitaammassa verkossa, kuten internetin välityksellä, pelattavuus kärsii. (Gambietta 2014.)

Pohditaan tätä ongelmaa esimerkiksi autopelin avulla. Kuvitellaan, että paketin lähettäminen asiakkaalta palvelimelle kestää 100 ms. Pelaajan painaessa esimerkiksi auton kääntämiseen käytettävää näppäintä, palvelin saa tämän tiedon siis 100 ms myöhemmin. Kun palvelin käsittelee tiedon ja lähettää auton kää-

tymiseen liittyvän paketin takaisin pelaajalle, oletetaan, että paketin lähettäminen kestää jälleen 100 ms. Tällöin pelaajan ohjaama auto kääntyy vasta 200 ms näppäimen painalluksen jälkeen. 200 ms viive ei välttämättä kuulosta suurelta, mutta huono responsiivisuus tekee kuitenkin tällaisen nopeatempoisen pelin pelattavuudesta huonon. (Gambietta 2014.)

### **3.6.3 Asiakaspään ennakointi**

Eräs tapa, jolla saa vähennettyä latenssin aiheuttamaa negatiivista vaikutusta, on niin sanottu Client-side prediction, eli asiakaspäässä tapahtuva ennakointi. Tällä tavalla kaikki pelin tapahtumat käsitellään edelleen palvelimella, mutta pelaajan antaessa syötteitä esimerkiksi liikuttaakseen pelihahmoa, pelihahmo liikkuu pelaajan peliruudulla ilman viivettä. Samalla syötteet lähetetään palvelimelle, jossa ne käsitellään ja lähetetään pelaajille. Jos asiakaspäässä tehty ennakointi poikkeaa palvelimelta takaisin tulleista tiedoista, pelihahmon tietoihin tehdään korjaus. Hyvänä puolena pelaajan ohjaama hahmo toimii näin hyvin responsiivisesti, mutta latenssin ollessa korkea, ero palvelimelta tuleviin tietoihin voi olla hyvinkin suuri aiheuttaen esimerkiksi pelihahmon äkillisen hyppäämisen toiseen paikkaan. Tähänkin ongelmaan on kuitenkin olemassa keinoja, joiden avulla ennakoinnista saadaan paremmin toimiva ja korjauksesta sulavampi. (Bernier 2001.)

### **3.6.4 Dead reckoning**

Kun pelataan monen pelaajan verkkopeliä, pelaajien tietojen lähettäminen toisille pelaajille tuo omat haasteensa. Kun asiakkaalta lähetetään paketti, joka sisältää tietoja esimerkiksi pelaajan positiosta, nopeudesta, kiihtyvyydestä, rotaatiosta ynnä muusta, paketin perille pääsy muille asiakkaille palvelimen kautta kestää jonkin aikaa. Muiden pelaajien liikkeitä pystytään kuitenkin ennakoimaan paketissa lähetettyjen tietojen avulla. Tällöin esimerkiksi toisen pelaajan ohjaama auto jatkaa matkaansa ylläpitäen samaa suuntaa, nopeutta ja kiihtyvyyttä kuin viimeksi saapuneessa paketissa on määritelty, kunnes seuraava paketti

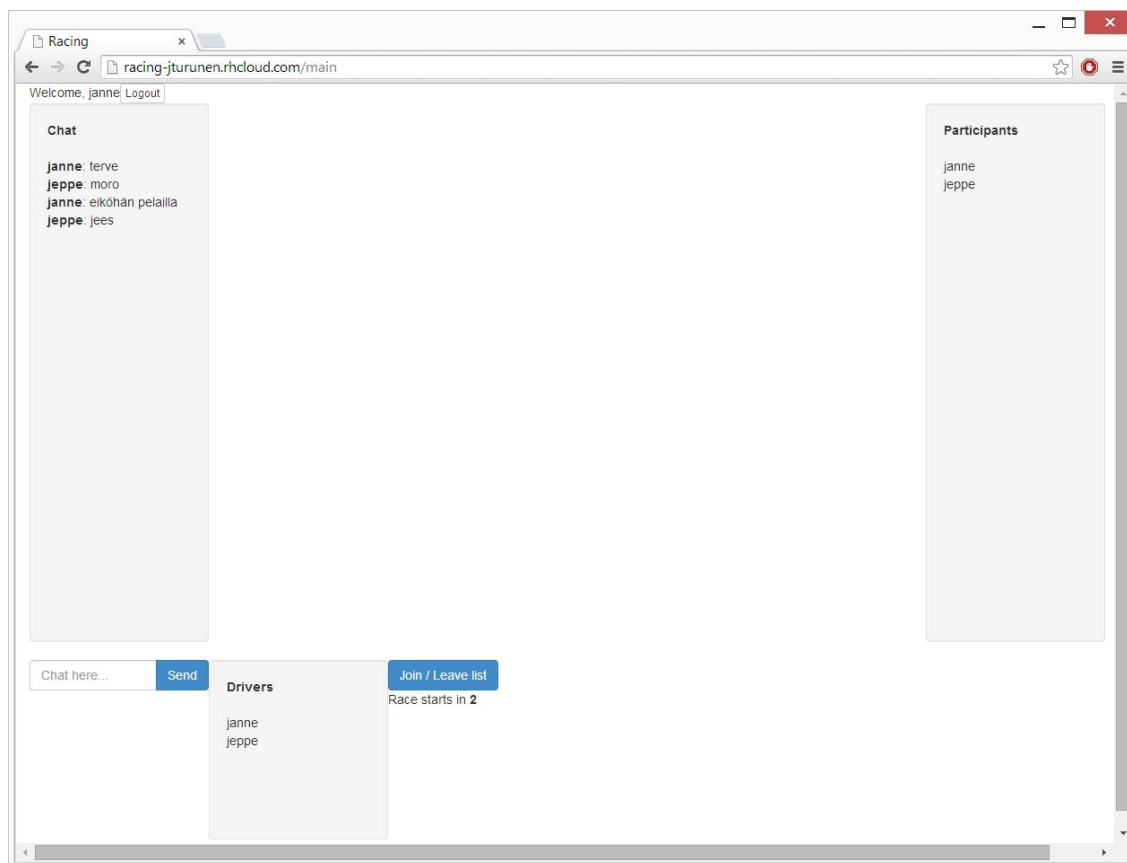
saapuu ja asettaa uudet arvot. Tällaisessa tapauksessa ongelmaksi muodostuu se, jos toinen pelaaja tekee äkillisen suunnanmuutoksen ja uuden paketin saapuesssa muille pelaajille hänen ohjaama autonsa onkin reilusti eri paikassa kuin ennakointi. Tällöin pelaajan auton tiedot korjaantuvat muilla pelaajilla ja auto hyppää oikeaan paikkaan, mikä ei tietenkään näytä hyvältä. Tähän ongelmaan on kuitenkin apu interpoloinnista, jolloin kanssapelaajan ennakointi tehdään aikaisempien pakettien tietojen perusteella. (Tan 2011.)

### **3.7 Verkkosivuston kehittäminen ja toiminta**

Projektissa tavoitteena oli saada tehdyksi sivusto, jonka ei tarvinnut olla ulkoisesti näyttävä. Pääasia tässä vaiheessa oli saada sivusto toimivaksi kokonaisuudeksi. Sivustoon tuli etusivun lisäksi vain kirjautumis-, rekisteröitymis- ja pääsivu, jossa peliin liitytään ja peliä pelataan ja jossa on mukana myös pelaajien keskustelualue.

Kuvassa 16 on kuvankaappaus sivustolle kehittelemästani yksinkertaisesta pääsivusta. Sivulla on mukana keskustelualue, kirjautuneiden käyttäjien alue sekä alue, johon voi liittyä käyttäjän halutessa peliin mukaan. Kun käyttäjä liittyy ajajien listaan, hänen socketinsa liitetään odotushuoneeseen ja palvelin aloittaa laskurin, jonka saavuttaessa nollan peli käynnistyy. Pelin käynnistyessä kaikki käyttäjät, jotka ovat sillä hetkellä ajajien listalla mukana, saavat palvelimelta viestin pelin käynnistymisestä ja heidät siirretään pelitilaan.





Kuva 16. Sivuston lobby, jossa on chat ja jossa pelit järjestetään.

Itse peliruudulle on määritelty elementti keskusteluosion ja käyttäjäosion väliin ja peliruutu ilmestyy vain niille käyttäjille, jotka peliin olivat liittyneet. Kun peli on käynnistynyt, muut käyttäjät voivat sen jälkeen liittyä ajajien listalle ja aloittaa toisen pelin halutessaan. Jokaiselle pelille palvelin arpoo summittaisen pelihuone-numeron ja kyseisessä pelihuoneessa olevat pelaajat lähettävät pelitietoja vain samassa huoneessa oleville sotkematta muita käynnissä olevia pelejä.

Sivuston tyylin määrittämisessä käytin apuna Twitter Bootstrap -kehystä. Bootstrapin avulla verkkosivustosta saa helposti responsiivisen tehden verkkosivuista käytettävämmän mobiililaitteilla. Koska tässä projektissa kehittelin tietokoneelle suunnattua peliä, otin responsiivisuuden pois käytöstä ja käytin vain Bootstrapin tarjoamia tyylejä hyväkseni.



### 3.8 Pelin ja sivuston julkaisu

Halusin saada pelini julkaistua verkossa ja päädyin OpenShift-pilvialustaan. OpenShift tukee Node.js:ää ja sivuston perustaminen ja sovelluksen siirtäminen sinne vaikutti helpolta operaatiolta. Opinnäytetyössä kehitetty pelin julkaisu tapahtuu GPLv3-lisenssin alaisena.

#### 3.8.1 OpenShift-pilvipalvelu

OpenShift on Red Hat -yhtiön ylläpitämä alusta, jonka avulla ohjelmistokehittäjät pystyvät kehittämään ja ylläpitämään sovelluksia pilviympäristössä. OpenShift tarjoaa erilaisia versioita käytettäväksi. OpenShift Online on julkisessa pilvessä toimiva alusta, joka tukee muun muassa Java-, Ruby-, PHP-, Node.js-, Python- ja Perl-ohjelmointikieliä. OpenShift Enterprise soveltuu yrityksille, jotka haluavat ajaa sovelluksia yksityisessä pilvessä tai omilla palvelimilla ja OpenShift Origin on tarkoitettu sellaisille, jotka haluavat osallistua OpenShiftin avoimen lähdekoodin projektiin. (OpenShift 2014a.)

OpenShift Onlinesta on saatavilla ilmaisia ja maksullisia versioita. Ilmaiseen versioon saa laitettua korkeintaan kolme pientä sovellusta ja ominaisuudet riittävät erinomaisesti pienen projektin kehittämiseen, kuten tässä opinnäytetyössä kehiteltyyn sivustoon ja autopeliin. Ilmaisessa versiossa jokaiselle tällaiselle pienelle sovellukselle on varattuna 512MB keskusmuistia ja 1GB levytilaa. Jos tarvetta on saada tilaa useammalle tai suuremmille sovelluksille, joutuu siirtymään maksulliseen Bronze- tai Silver-jäsenyyteen. Kuvassa 17 näkyy jäsenyyksiin liittyviä hintaeroja. Yksi gear koostuu aina sovelluksesta ja siihen liitetystä osista, joita voi olla esimerkiksi MongoDB-, MySQL- tai PostgreSQL-tietokanta tai RockMongo-työkalu MongoDB-tietokannan ylläpitämistä varten. Palveluissa on myös muita eroja esimerkiksi tukiasoissa. (OpenShift 2014b.)

 USD  CAD  EUR	Free	Bronze	Silver
BASE PRICE	Free	€0 / month	€15 / month
 GEARS			
Small gears (1-3)	Free	Free	Free
Small gears (4+)	✖	€0,02 / hour	€0,02 / hour
Medium gears	✖	€0,04 / hour	€0,04 / hour
Large gears	✖	€0,08 / hour	€0,08 / hour
Gear maximum	3	16	16+
Gear idling	2 days	None	None
 STORAGE			
Base storage (per gear)	1GB	1GB	6GB
Additional storage	✖	€1,00 / GB / month	€1,00 / GB / month

Kuva 17. OpenShift Onlinen versioiden hintaeroja (OpenShift 2014b).

OpenShiftiin luotu sovellus saa automaattisesti domainin, joka on muotoa [sovelluksen nimi]-[käyttäjän nimi].rhcloud.com, mutta myös ilmaisessa versiossa sovelluksen voi siirtää omaan domainiin ilman lisämaksuja.

## 4 Pohdinta

Ennen opinnäytetyötäni minulla ei ollut minkäänlaista kokemusta verkkoselaimella pelattavien pelien tekemisestä eikä verkossa pelattavista moninpeleistä. JavaScript-ohjelmointikielestä minulla oli myös aika vähän kokemusta, joten tähän projektiin lähtiessäni se oli kuin hyppy tuntemattomaan.

Ennen projektiin ryhtymistä en ollut koskaan ajatellut, kuinka haastavaa reaaliaikaisen nopeatempoisen verkkopelin tekeminen onkaan. Verkkoviiveet aiheuttavat sen, että pelin saaminen pelattavuudeltaan hyväksi joutuu käyttämään monia erilaisia keinoja hyvän lopputuloksen saavuttamiseksi. Valitettavasti en päässyt käytännössä kunnolla testaamaan keinoja parantaa latenssin aiheutta-

mia ongelmia, koska ainakaan käytössä ollut Phaserin versiota ei saanut käytettyä palvelinpuolella lainkaan.

Sivuston perustaminen ja sen lisääminen pelin kanssa OpenShift-palveluun toimi suhteellisen mutkattomasti. Kokemukseni perusteella tulen varmasti tulevaisuudessakin käyttämään kyseistä palvelua ja kehittämään sinne mahdollisesti myös jotain muuta.

Mielestäni onnistuin tavoitteissani kohtalaisen hyvin. Sain perustettua yksinkertaisen sivuston, jossa on toimiva chat käyttäjille sekä lisättyä sivustolle pelin, joka suppeasta sisällöstään huolimatta toimii kohtalaisen hyvin.

#### **4.1 Kehittämisideat**

Opinnäytetyön aikana aikaansaamani prototyyppi on sisällöltään hyvin suppea, mutta jatkokehitysmahdollisuudet ovat mielestäni hyvät. Prototyypissä on vain yksi ajoneuvomalli ja yksi rata. Suhteellisen pienellä työmäärällä peliin saisi lisättyä erilaisia ajoneuvoja erilaisin ominaisuuksin sekä rakennettua erilaisia ratoja ajettavaksi. Ajoneuvoille voisi myös lisätä viritysmahdollisuuksia. Ajoneuvoihin voisi esimerkiksi asentaa aseita, tehokkaampia moottoreita, parempia renkaiden ja muita suorituskykyä parantavia osia. Uusien ajoneuvojen hankkimiseen ja niiden virittämiseen voisi mahdollisesti liittää myös mikromaksut.

Tekoälyn tekeminen peliin toisi lisää monipuolisuutta. Tekoäly voisi yksinkertaisesti olla sellainen, että rataa määritellään näkymättömiä pisteitä, joita kohti tietokoneen ohjaama auto kiihdyttää. Jokaiselle pisteelle on erikseen määriteltävä oma suorakaiteen muotoinen alue ja tekoälyauton osuessa tällaiseen suorakaiteeseen, se lopettaa kyseistä pistettä kohti ajamisen ja lähtee ajamaan kohti seuraavaa pistettä. Oikeanlaisella pisteiden ja suorakaiteiden sijoittamisella voidaan saada yksinkertainen, mutta hyvin kilpailukykyinen tekoäly peliin mukaan.

Jotta moninpelin saisi toimivammaksi, pelin koodia tulisi pystyä ajamaan palvelinpuolella. Ainakaan projektissa käytössä ollut Phaserin versio ei toimi palveli-

mella. Phaser vaatii canvas-elementin toimiakseen, joten palvelinpuolella Phaseria ei ainakaan sellaisenaan pysty käyttämään. Tästä syystä prototyypistä puuttuu vastustajien autoihin törmäily. Jos olisin yrittänyt saada autojen törmäilyt mukaan, minun olisi itse pitänyt tehdä palvelinpuolelle jonkinlainen törmäyksen tunnistus. Ajan puutteen vuoksi en tähän ainakaan tässä vaiheessa kuitenkaan ryhtynyt.

Peliin saisi monipuolisuutta lisäämällä siihen erilaisia kilpailumuotoja. Pelaajat voisivat ajaa toisiaan vastaan yksittäisten kisojen lisäksi esimerkiksi jonkinlaisissa liigoissa tai cupeissa, joissa ajettaisiin useampia kisoja eri radoilla ja annetaan ajajille pisteitä sijoitusten mukaan. Prototyypistä puuttuu myös äänet, joten se on yksi ensimmäisistä asioista, jonka jatkokehityksessä voisi peliin lisätä.

## Lähteet

- Bernier, Y. 2001. Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization  
[https://developer.valvesoftware.com/wiki/Latency\\_Compensating\\_Methods\\_in\\_Client/Server\\_In-game\\_Protocol\\_Design\\_and\\_Optimization](https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization). 5.5.2014.
- Cantelon, M., Harter, M., Holowaychuk, T.J. & Rajlich, N. 2013. Node.js in Action. New York: Manning.
- DreamersLab. 2011. npm basic commands.  
<http://dreamerslab.com/blog/en/npm-basic-commands/>. 26.2.2014.
- Emaze. 2013. The Benefits of HTML5 vs. Adobe Flash.  
<http://www.emaze.com/blog/html5-vs-flash/>. 15.2.2014.
- Gambietta, G. 2014. Fast-Paced Multiplayer (Part I): Introduction  
<http://www.gabrielgambetta.com/fpm1.html>. 3.5.2014.
- GIMP. 2013. About GIMP.  
<http://www.gimp.org/about/introduction.html>. 5.5.2014.
- Google Developers. 2012. Chrome V8 Introduction.  
<https://developers.google.com/v8/intro>. 7.4.2014.  
<http://www.emaze.com/blog/html5-vs-flash/>. 15.2.2014.
- Harris, A. 2014. The Birth of Node: Where Did it Come From? Creator Ryan Dahl Shares the History.  
<http://devopsangle.com/2013/04/01/the-birth-of-node-where-did-it-come-from-creator-ryan-dahl-shares-the-history/>. 16.2.2014.
- McCarthy, K. 2011. Node.js Interview: 4 Questions with Creator Ryan Dahl.  
<http://bostinno.streetwise.co/2011/01/31/node-js-interview-4-questions-with-creator-ryan-dahl/>. 16.2.2014.
- MongoDB. 2013. Agile and Scalable.  
<https://www.mongodb.org/>. 23.3.2014
- MongooseJS. 2014.  
<http://mongoosejs.com/index.html>. 23.3.2014
- MongooseJS guide. 2014.  
<http://mongoosejs.com/docs/guide.html>. 2014
- NodeJS. 2014a.  
<http://nodejs.org/>. 15.2.2014.
- NodeJS 2014b.  
<http://nodejs.org/industry/>. 5.5.2014.
- NodeJS blog. 2011a. Porting Node to Windows With Microsoft's Help.  
<http://blog.nodejs.org/2011/06/23/porting-node-to-windows-with-microsoft%25e2%2580%2599s-help/>. 29.4.2014.
- NodeJS blog. 2011b. Node v0.6.3.  
<http://blog.nodejs.org/2011/11/25/node-v0-6-3/>. 26.2.2014.
- NpmJS. 2014. package.json.  
<https://www.npmjs.org/doc/json.html>. 27.2.2014.
- OpenShift. 2014a. OpenShift Online.  
<https://www.openshift.com/products/online>. 25.3.2014.
- OpenShift. 2014b. OpenShift Online Pricing.  
<https://www.openshift.com/products/pricing>. 26.4.2014.
- Passport. 2013. Overview.  
<http://passportjs.org/guide/>. 20.3.2014.

- Phaser. 2014. Desktop and Mobile HTML5 game framework.  
<http://phaser.io/>. 23.3.2014.
- Phaser Docs. 2014a. Class: RenderTexture.  
<http://docs.phaser.io/Phaser.RenderTexture.html>. 5.5.2014.
- Phaser Docs. 2014b. Class: Emitter.  
<http://docs.phaser.io/Phaser.Particles.Arcade.Emitter.html>. 5.5.2014
- Phaser Docs. 2014c. Class: SoundManager.  
<http://docs.phaser.io/Phaser.SoundManager.html>. 29.4.2014.
- Phaser General Documentation. 2014a. Game.  
<https://github.com/photonstorm/phaser/wiki/Phaser-General-Documentation--Game>. 7.4.2014.
- Phaser General Documentation. 2014b. States.  
<https://github.com/photonstorm/phaser/wiki/Phaser-General-Documentation--States>. 3.5.2014.
- Photon Storm. 2013. Tutorial: Making your first Phaser game.  
<http://www.photonstorm.com/phaser/tutorial-making-your-first-phaser-game>. 7.4.2014.
- Rogers, C. 2013. Web Audio API.  
<http://www.w3.org/TR/webaudio/#APIOverview>. 25.4.2014.
- Slicks 'n Slide. 1993. Suomi: Kauppinen, T.
- Tan, M. 2011. Dead Reckoning.  
<http://zantheinvisible.blogspot.fi/2011/04/dead-reckoning.html>.  
 5.5.2014
- W3resource. 2013. Introduction to Node.js.  
<http://www.w3resource.com/node.js/node.js-tutorials.php>. 16.2.2014.
- W3schools. 2014. HTML5 Audio.  
[http://www.w3schools.com/html/html5\\_audio.asp](http://www.w3schools.com/html/html5_audio.asp). 26.4.2014.
- WebGL wiki. 2011. Getting Started.  
[http://www.khronos.org/webgl/wiki/Getting\\_Started](http://www.khronos.org/webgl/wiki/Getting_Started). 23.3.2014.